Software-Reengineering — Werkzeuge und Prozesse

Andreas Winter
Universität Koblenz-Landau
www.uni-koblenz.de/~winter/
winter@uni-koblenz.de

Zusammenfassung

Dieser Beitrag fasst kurz die Aktivitäten der Forschungsgruppe Softwaretechnik der Universität Koblenz im Bereich des Software-Reengineerings zusammen. Diese Aktivitäten umfassen die Entwicklung verschiedener Werkzeuge zur Unterstützung des Reengineerings, die Integration unterschiedlicher Werkzeuge zu umfassenderen Reengineering-Umgebungen und die Untersuchung der Aktivitäten und Prozesse des Software-Reengineerings.

1. Reengineering

Software Reengineering (kurz: Reengineering) umfasst alle Aktivitäten, deren Ziel die qualitative Verbesserung bestehender Softwaresysteme ist (vgl. [1]). Mit dieser *Verbesserung* ist i. Allg. eine Verlängerung des Softwarelebens intendiert. Reengineering kann als *Transformation* aufgefasst werden, durch die ein Softwaresystem, bestehend aus ausführbaren Programmen, Quelltexten und zugehöriger Dokumentation, in ein neues, „reengineertes“ Softwaresystem überführt wird.

Reengineering Maßnahmen werden mit unterschiedlichen Zielsetzungen durchgeführt. Diese umfassen u. A. Maßnahmen

- zur *(korrektiven) Wartung*, d. h. zur unverzüglichen Behebung von Fehlern,
- zur (Release-getrieben) *korrektiven Systemerhaltung*, d. h. zur Behebung von Fehlern in der nächsten ausgelieferten Version,
- zur *Erweiterung/Adaption*, d. h. zur Anpassung an geänderte Anforderungen,
- zur *Migration*, d. h. zur Überführung von Softwaresystemen in andere Zielumgebungen,
- zur *Integration*, d. h. zur Kombination verschiedener Softwaresysteme,
- zur *Sanierung*, d. h. zur Verbesserung der Softwarequalität (ohne Änderung der Funktionalität),
- zur *Re-Dokumentation*, d. h. zur nachträglichen Erstellung von Dokumentationen, und
- zur *Ablösung*, d. h. zur Vorbereitung der Außerdienststellung eines Softwaresystems (vgl. auch [8]).

Diese unterschiedlichen Reengineering-Maßnahmen erfordern unterschiedliche Vorgehensweisen mit unterschiedlicher technischer Unterstützung. Forschungsarbeiten der Arbeitsgruppe Softwaretechnik der Universität Koblenz befassen sich mit der Entwicklung von anpassbaren Reengineering Werkzeugen (Kap. 2), der Kombination solcher Werkzeuge (Kap. 3) und der Erhebung, Analyse und Definition von Vorgehensweisen in Reengineering-Prozessen (Kap. 4).

2. Werkzeuge

Eine zentrale Voraussetzung aller Reengineering Maßnahmen ist das grundlegende Verstehen und Nachvollziehen der vorhandenen Softwaresysteme. *GUPRO (Generische Umgebung zum Programmverstehen)* [2] bietet hierzu ein anpassbares Werkzeug zur Quelltextanalyse und Visualisierung.

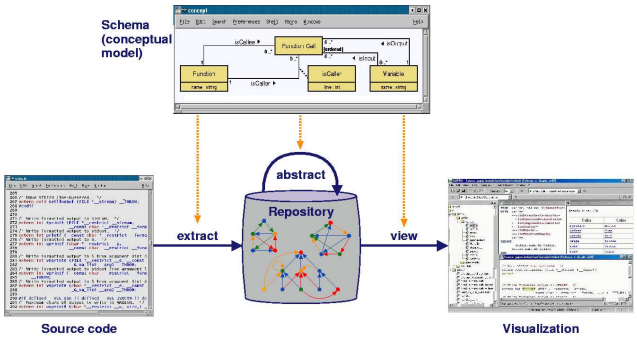

Abbildung 1. GUPRO-Architektur

GUPRO folgt der *Extract-Abstract-View-Metapher* [6]. Fakten über den Quelltext werden durch geeignete Parser aus dem Quelltext entnommen (Extract) und in einem einfach und effizient analysierbaren Graphen-Repository abgelegt. Diesen Daten werden mittels geeigneter Analyse-Werkzeuge ausgewertet (Abstract) und anschließend werden die Analyse-Ergebnisse in geeigneter Form visualisiert (View). Abbildung 1 zeigt die grundlegende GUPRO-Architektur.

Mit GUPRO können beliebige Informationen über Programmtexte im Repository abgelegt werden, die mit Hilfe einer generellen Auswertungsmaschine analysiert werden können. Die Anpassung von GUPRO an konkrete Reengineering Fragestellungen erfolgt durch Entwicklung einer *problemangemessenen Repository-Struktur* (vgl. Abb. 1, der Bereitstellung *geeigneter Extraktoren* zur Überführung der Quelltexte in dieses Repository und der Konfektionierung der Analysemaschine durch Formulierung

von *Graph-Anfragen* [4] an das Repository oder durch Realisierung durch geeignete *Graph-Algorithmen*.

GUPRO-Werkzeuge wurden für verschiedene Fragestellungen der Programmanalyse entwickelt. So unterstützen GUPRO-Instanzen die *Architekturanalyse* mehrsprachiger Softwaresysteme (Cobol, PL/1, CSP, JCL, IMS-DB, SQL bzw. Java, C, C++, RDBMS) aus dem Finanz- und Versicherungsbereich und die *feingranualare Analyse* von C- und Ada-Programmen auf Ebene der Syntaxgraphen im Rahmen von Sicherheitsanalysen und Softwarezertifizierung.

3. Interoperabilität

Reengineering-Werkzeuge werden für unterschiedliche Aufgaben in der Programmanalyse und in der Software-Weiterentwicklung erstellt. Hierbei werden einerseits Basiskomponenten mehrfach entwickelt, andererseits erfordert die Entwicklung und Weiterentwicklung umfangreicher Softwaresysteme die Integration verschiedener Werkzeuge zu leistungsstarken Software-Entwicklungsumgebungen.

Grundlage der Interoperabilität dieser Werkzeuge ist der *standardisierte Datenaustausch* zwischen den einzelnen Komponenten. Reengineering-Werkzeuge basieren in der Regel auf relationalen oder Graph-basierten Datenstrukturen. Zum Austausch solcher Daten wurde gemeinsam mit Richard C. Holt (University of Waterloo), Susan Elliott Sim (University of California, Irvine), und Andy Schürr (TU Darmstadt) die *Graph Exchange Language (GXL)* [3, 9] entwickelt.

GXL ist eine XML-basierte Sprache, die für den Austausch unterschiedlicher Graphstrukturen (Bäume, gerichtete Graphen, attributierte Graphen, hierarchische Graphen, Hypergraphen, etc.) konfektioniert werden kann. In GXL werden sowohl Graphen zur Beschreibung der Graphstruktur (Graphschemata), als auch Instanzgraphen unabhängig von ihrer konkreten Struktur durch gleichartige Dokumente ausgetauscht.

GXL wurde auf dem Dagstuhl-Seminar „Interoperability of Reengineering Tools“ im Januar 2001 als Standardaustauschformat im Reengineering ratifiziert. GXL hat inzwischen Einzug in diverse weitere Graph-bezogener Austauschformate gefunden und wird von mehr als 40 Werkzeugen unterstützt.

4. Reengineering Prozesse

Heute übliche Vorgehensmodelle zur Software-Entwicklung beschränken sich in erster Linie auf die Neuentwicklung von Softwaresystemen. Reengineering-Aktivitäten sind in Vorgehensmodellen der Softwaretechnik kaum berücksichtigt [7, 5].

Aktuelle Arbeiten beschäftigen sich mit der Identifizierung und Kategorisierung unterschiedlicher Reengineering-Maßnahmen (vgl. Kap. 1) und der Erhebung der hierbei benötigten und durchgeführten Aktivitäten. Ziel ist die Entwicklung und Erprobung von Referenz-Prozessmodellen zur Wartung und Weiterentwicklung von Softwaresystemen, einschließlich der hierzu benötigten Unterstützung durch Reengineering-Werkzeuge.

5. Ausblick

Software Reengineering-Aktivitäten nehmen eine immer wichtiger werdende Rolle in der Entwicklung von Softwaresystemen ein. Isolierte Neuentwicklungen von Softwaresystemen finden kaum noch statt. Es dominiert die *Evolution* bestehender Systeme. Die Evolution bestehender Softwaresysteme erfordert ein Überdenken der Vorgehensmodelle zur Softwareerstellung und die Entwicklung von Methoden, Techniken und Werkzeugen zur Unterstützung.

Praktiker aus der Softwareindustrie und Wissenschaftler aus der Softwaretechnik sind hier gefordert *gemeinsam* Reengineering-Methoden und Werkzeuge zur Verbesserung der Software-Evolution zu entwickeln. Ein Forum hierzu bietet die *GI Fachgruppe Reengineering* (in Gründung) (`http://mailhost.uni-koblenz.de/mailman/listinfo/reengineering`) und der jährlich ausgerichtete *Workshop Software Reengineering* (`http://www.uni-koblenz.de/~ist/wsr/`).

Literatur

[1] J. Bergey, W. Hefley, W. Lamia, and D. Smith. A Reengineering Process Framework. Software Engineering Institute, Carnegie Mellon University, 1995.

[2] J. Ebert, B. Kullbach, V. Riediger, and A. Winter. GUPRO – Generic Understanding of Programs, An Overview. *Electronic Notes in Theoretical Computer Science (`http://www.elsevier.nl/locate/entcs/volume72.html`)*, 72(2), 2002.

[3] R. C. Holt, A. Winter, and A. Schürr. GXL: Toward a Standard Exchange Format. In *7th Working Conference on Reverse Engineering*. IEEE Computer Society, Los Alamitos, 162–171. 2000.

[4] M. Kamp and B. Kullbach. GReQL - Eine Anfragesprache für das GUPRO–Repository, Sprachbeschreibung. Projektbericht 8/2001, Universität Koblenz-Landau, Institut für Softwaretechnik, Koblenz, August 2001.

[5] U. Kuhlmann and A. Winter. Softwarewartung und Prozessmodelle in Theorie und Praxis. *Softwaretechnik-Trends*, 24(2):37–38, Mai 2004.

[6] C. Lange, H. Sneed, and A. Winter. Comparing Graph-based Program Comprehension Tools to Relational Database-based Tools. In *9th International Workshop on Program comprehension*. IEEE, Los Alamitos, 209–218. 2001.

[7] S. Opferkuch and J. Ludewig. Software-Wartung, Eine Taxonomie. *Softwaretechnik-Trends*, 24(2):35–36, Mai 2004.

[8] H. M. Sneed, M. Hasitschka, and M.-T. Teichmann. *Software-Produktmanagement - Wartung und Weiterentwicklung bestehender Anwendungssysteme*. dPunkt, erscheint 2004.

[9] A. Winter. Exchanging Graphs with GXL. In P. Mutzel, M. Jünger, and S. Leipert, editors. *Graph Drawing*, LNCS 2265. Springer, Berlin, 485–500. 2002.